applied sciences

MDPI

Article

The Derivation of Defect Priorities and Core Defects through Impact Relationship Analysis between Embedded Software Defects

Sang Moo Huh [1] and Woo-Je Kim [2,*]

[1] Graduate School of Public Policy and Information Technology, Seoul National University of Science and Technology, Gongneung-ro 232, Nowon-gu, Seoul 01811, Korea; norhuh@naver.com
[2] Department of Industrial Engineering, Seoul National University of Science and Technology, Gongneung-ro 232, Nowon-gu, Seoul 01811, Korea
* Correspondence: wjkim@seoultech.ac.kr; Tel.: +82-10-5471-0920

Received: 4 September 2020; Accepted: 2 October 2020; Published: 4 October 2020

Abstract: As embedded software is closely related to hardware equipment, any defect in embedded software can lead to major accidents. Thus, all defects must be collected, classified, and tested based on their severity. In the pure software field, a method of deriving core defects already exists, enabling the collection and classification of all possible defects. However, in the embedded software field, studies that have collected and categorized relevant defects into an integrated perspective are scarce, and none of them have identified core defects. Therefore, the present study collected embedded software defects worldwide and identified 12 types of embedded software defect classifications through iterative consensus processes with embedded software experts. The impact relation map of the defects was drawn using the decision-making trial and evaluation laboratory (DEMATEL) method, which analyzes the influence relationship between elements. As a result of analyzing the impact relation map, the following core embedded software defects were derived: hardware interrupt, external interface, timing error, device error, and task management. All defects can be tested using this defect classification. Moreover, knowing the correct test order of all defects can eliminate critical defects and improve the reliability of embedded systems.

Keywords: embedded software defects; content analysis; DEMATEL; cause–effect relationship; core defects; impact relationship analysis

1. Introduction

Embedded systems are used in various industries, including automotive, railway, construction, medical, aerospace, shipbuilding, defense, and space. However, these systems have software defects that can cause fatal accidents. In the medical field, a safety-critical system defect in radiotherapy resulted in more than six human injuries caused by excessive radiation in two years [1]. In the space sector, the Ariane 5 Flight 501 that failed its maiden flight is reportedly an example of an accident caused by a software defect [2]. In the defense sector, these defects caused the deaths of 28 US Army soldiers and left 98 injured when a Patriot missile malfunctioned in Dhahran, Saudi Arabia [3]. As such, fatal consequences can occur if defects are not eliminated in embedded systems. Therefore, all defects must be collected and classified so that no untested embedded defects exist. As a result of investigating an embedded software defect study, many studies based their analyses on embedded architecture, as well as defects in interface, dynamic memory, and exception handling, and other miscellaneous defects found in airplane or space-exploration applications. Although these studies are meaningful in each field, all defects have not been consolidated and classified. If defects from some previous studies

are the only ones referenced and tested, there may be defects that have not been tested and cause serious problems.

To solve such problems, this study collected all possible defects experienced globally and classified them into a systematic and integrated view by applying a content analysis technique. In addition, when a defect occurs, the defect can affect other defects. Due to this characteristic, the defect can become more and more serious. In this paper, these defects are considered core defects because they can cause critical failures in the embedded system. These defects were derived by analyzing the impact relationship between embedded software defects and applying decision-making trial and evaluation laboratory (DEMATEL) methods. Using these methods will enable developers to eliminate defects without omission using integrated defect types and improve embedded software quality via the intensive management of core defects.

Various defects in the pure software field have been collected, classified, and studied worldwide [4,5]. Mäntylä and Lassenius [4] argue that code review often undermines the benefits of core review by focusing on the number of defects instead of the defect type. For this reason, they collected and categorized defects that are useful in code review. Huh and Kim [5] collected a series of pure software defect studies (Table 1) and classified the defects into specific functional categories in their meta-analysis (Table 2). Using the analytic network process (ANP), they could derive what they identified as core defects in general software applications, such as personnel, salary, and accounting systems. By analyzing the impact relationship of those pure software defects, they could derive a set of core defects. In their study, they concluded that targeting core defects could eliminate any related peripheral defects, making it a more efficient troubleshooting method. The present study expands on Huh and Kim's [5] defect-classifying study, using their list of pure software defects as the present study's list of pure software area defects.

Table 1. Huh and Kim's [5] references for pure software defects.

Researchers	Software Defects
IEEE 1044 [6]	IEEE 1044-1993 standard classification for software anomalies
HP Huber JT [7]	Hewlett Packard's defect origins, types, and modes
IBM ODC [8]	Orthogonal defect classification (ODC) for software design and code
Other researchers	Collected software defects defined by 21 researchers worldwide

Table 2. Huh and Kim's [5] classified list of pure software defects.

Category	Defects	Sub-Defects
Logic	Conditional statement	Checking, duplicating IF statement, empty IF statement, compared with other variables, missing important conditions (case, etc.)
	Rotation logic	Infinite loops, infinite recursions, algorithm, logic sequences, flow control, error checking, check scope, status handling, missing a step
	Concurrent logic	Synchronization, race conditions, mutual exclusions, critical sections, concurrent processing, coordination process, condition loads
Interface, Timing	External interface	Human interface error, different protocols, incorrect protocols
	Wrong function (internal interface)	Return pointers, incorrect Application programming interface(API), software architecture, function/class/object relationship, no existence subroutines, missing return values, incorrect parameter's error, calls incorrect subroutines, calls incorrect module, incorrect interrupt
	I/O timing	Time overrun, incorrect Input/Output(I/O) timing
Computation	Division by zero	Divide by zero
	Expression	Wrong operator, incorrect parenthesis usage, different unit calculations, incorrect sign usage, missing expressions, wrong expressions
	Precision loss	Mixed modes, round/truncation calculations, underflow, overflow
Data	Data structure	Error of data design, wrong data structures, wrong data units, pack/unpack
	Data usage	Leaks, use after free, un-assignment, initialization pointer, memory violation, other variable type usages, other variable use dimensions, null pointers, wrong index, use other flags, error script variable usage, save/access errors, initialization errors
	Data value self	Wrong input data, wrong operation data, wrong external data, wrong sensor data

Next, the present study investigated other studies that focused on embedded software defects. Barr [9,10] presented 10 important embedded software defects: race conditions, non-reentrant functions, missing volatile keywords, stack overflow, heap fragmentation, memory leaks, deal locks, priority inversions, and incorrect priority assignments. On the other hand, Lutz [11] classified 387 errors encountered during the Voyager and Galileo missions. Meanwhile, Hagar [12] presented test methods for various embedded defects. Lee et al. [13] defined 11 faults for input, control, and output, and then tested them within a vehicle's embedded system. Jung et al. [14] studied defects that violate the Motor Industry Software Reliability Association-C (MISRA-C) 2004 coding rules, using static analysis tools. Then, Bennett and Wennberg [15] tried a different approach to defect analysis by studying a method of cost-effective testing using an integrated test for five types of defects found during spacecraft development. Researchers like Seo [16] studied and tested defects that occur in the interface between the software (SW) and hardware (HW) of embedded systems. Choi [17] defined dynamic memory defects and subsequently tested for them in embedded systems. Studies, such as Lee's [18] 2010 study, went as far as examining methods for recovering faults through exception processing routines when they occurred in embedded systems. Other researchers still approached their analysis by manually injecting and testing the defects, such as Cotroneo et al. [19], who investigated their test's ability to inject defects into an embedded system. Lee et al. [20] and Lee and Park [21] conducted similar fault injection tests, putting six defects into the defense embedded system. Lee [22] then also studied fault injection tests for six orthogonal defect classification (ODC) defects in an aerospace embedded system.

Despite the exhaustive number of studies conducted, they have neither comprehensively aggregated these defects nor classified them as mutually exclusive and collectively exhaustive (MECE). Moreover, there has been no attempt to derive significant defects using the influence relationship of the defects. For this reason, this study collected globally embedded software defects and classified them as mutually exclusive and collectively exhaustive(MECE), with the intent to derive core defects so that they can be applied to embedded software applications.

2. Materials and Methods

All collected defects were categorized as MECE to address numerous unique defects noted by the different researchers. This study used content analysis to categorize and integrate terms based on their characteristics and meanings [23], thus creating the categories used here. Then, the DEMATEL method was used to identify the impact relationship between the defect categories and distinguish cause defects from effect defects [24].

2.1. Content Analysis

First, the present study used content analysis, a method suited for studying multifaceted and sensitive phenomena and its characteristics, to categorize many defects [25]. This technique categorizes and structures information derived from textual material, quantifying qualitative data. However, the method can be time-consuming, and problems may arise when interpreting or transforming ambiguous or extensive information. Moreover, content analyses may suffer from researcher overinterpretation, calling into question the validity of the analysis [25]. However, with a step-by-step analysis, it is one way to effectively classify sensitive topics, with its constructed categories open to change whenever appropriate throughout the analysis process [26]. This mixed approach to data analysis enables researchers to measure the reliability of their classifications [27]. Generally, content analysis is performed using either Honey's content analysis technique or the bootstrapping technique. The present study applied the bootstrapping content analysis technique and utilized a seven-step procedure, as shown in Table 3 [23].

Following the bootstrapping procedures (Table 3), the researcher and collaborator classied each of the elements, respectively, and matched classifications are shown in Table 4. Researcher categories are recorded in the left column and collaborator categories are recorded in the top row. When there is a matched category for the category in the left column and the category in the top row, this is adopted

as an agreed category. If the categories do not match, new categories have to be created after discussion between the researcher and collaborators. The agreed categories were adopted through this repeated process of consensus. Additionally, elements are recorded within the diagonal cell of the agreed categories (elements are recorded as construct number). When the recorded elements in a diagonal cell match, they are then adopted as agreed elements. If any elements do not match, the researcher and collaborator reclassify through their discussion. The agreed elements were adopted through this repeated process of consensus. The reliability of the agreed elements is measured by the classification index, with reliability referring to the percentage of agreed values located diagonally. In this scale, 80%–89% or more is rated as good, while 90% or more is rated as excellent [23].

Table 3. The classification procedures of bootstrapping content analysis.

Step	Procedures
1	Create an appropriate category that contains the attributes of the first element
2	Create a new category if the following element is different from the first element
3	Distribute the following elements into similar categories
4	Combine and detach existing categories as needed to create new ones
5	Repeat until all elements are classified
6	Place all unclassifiable elements in the miscellaneous group
7	Repeat classification until the elements in the miscellaneous group account for less than 5% of the total

Table 4. An example of a reliability table where elements are matched by researchers and collaborators.

Researcher \ Collaborator	Category C1 (Matched with R1)	Category C2 (Matched with R2)	Category C3 (Non-Matched)
Category R1 (matched with C1)	1.1, 2.1, 4.2, 3.2, 3.3 (matched)	1.4 (non-matched)	-
Category R2 (matched with C2)	-	2.2, 2.3, 2.4, 3.1, 3.4 (matched)	2.5 (non-matched)
Category R3 (non-matched)		-	1.2, 4.3 (non-matched)
Category R4 (non-matched)	-	4.4 (non-matched)	4.1, 1.3 (non-matched)

2.2. DEMATEL Method

The decision-making trial and evaluation laboratory (DEMATEL) method was initially developed to solve complex and intertwined problems by the Science and Human Affairs Program of the Battelle Memorial Institute of Geneva. This study used the DEMATEL method for five reasons: (1) it can analyze the impact of relationships between complex factors; (2) it can create an impact relationship map (IRM) that can be used to visualize the relationship between factors, clearly illustrating one's effect on another; (3) the alternatives can be ranked, and these weights can be measured through a six-step derivation process to get the cause-and-effect relationships between elements [28]; (4) factors affected by other factors are assigned a lower priority, whereas factors that affect others are given higher priority [29]; and (5) lastly, a similar methodology was conducted by Seyed-Hossein et al. [30] with notably positive results, wherein they performed a reprioritization of the system failure modes by applying the DEMATEL method to the defects observed in the turbocharged engine. Their experiment covered the disadvantages of the traditional risk priority number (RPN) method for the failure mode and effects analysis (FMEA) defect of the said engine. This DEMATEL method, however, has two primary disadvantages: (1) the factors are only ranked according to the relationship between them, and (2) a relative weight cannot be assigned to each expert evaluation [31].

2.2.1. Step 1: Deriving the Direct Relation Matrix (DRM)

The impact values that the i row element affects the j column elements were collected from respondents, and the DRM was calculated, averaging the impacts values. The DRM A is shown below in Equation (1).

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1n} \\ \vdots & & \vdots & & \vdots \\ a_{i1} & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & & \vdots & & \vdots \\ a_{n1} & \cdots & a_{nj} & \cdots & a_{nn} \end{bmatrix} \quad (1)$$

2.2.2. Step 2: Normalizing the Matrix

The largest value or values are chosen by comparing the maximum value of the sum of rows to the maximum value of the sum of columns, seen in Equation (2). The DRM (A) is then divided by this value; then, the normalized matrix (N) is calculated, as seen in Equation (3).

$$s = max[max \sum_{j=1}^{n} A_{ji},\ max \sum_{i=1}^{n} A_{ij}] \quad (2)$$

$$N = \frac{A}{s} \quad (3)$$

2.2.3. Step 3: Calculating for the Total Relation Matrix (TRM, T)

The total influence matrix T is calculated by adding together all the direct and indirect effects using the normalized direct influence matrix N to get the TRM, T.

$$T = N + N^2 + N^3 + \cdots + N^m = N(I - N)^{-1}, \quad when\ m \to \infty \quad (4)$$

2.2.4. Step 4: Separating the Influencing (Cause) Elements and the Influenced (Effect) Elements

The sum of the rows (D) shows the level of direct influence. Meanwhile, the sum of the columns (R) represents the level of indirect influence, as seen in Equations (5) and (6). The D value numerically expresses the degree of how much one factor affects other factors, while the R-value expresses the degree of how much one factor is affected by other factors. On the one hand, D+R is the sum of the values affecting other factors and the values affected by other factors. On the other hand, D-R is the difference in the values that affect other factors and the value affected by other factors. The larger the value of D-R, the greater the influencing power of the factors; the smaller its value, the more it is affected by other factors [32]. The factors with positive D-R values are considered the cause group, while the elements with negative D-R values are considered the effect group [25].

$$D = [D_i]_{n\times 1} = [\sum_{i=1}^{m} T_{ij}]_{n\times 1} \quad (5)$$

$$R = \left[R_j\right]_{1\times n} = [\sum_{i=1}^{n} T_{ij}]_{n\times 1} \quad (6)$$

2.2.5. Step 5: Calculating the Threshold

The threshold is calculated as the average of the matrix, as seen in Equation (7) [28].

$$\sigma = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} [T_{ij}]}{n} \quad (7)$$

2.2.6. Step 6: Drawing the Cause and Effect Diagram

The cause and effect diagrams visualize the complex interrelationships of all elements and provide information on the most important elements and influencing factors [33]. The diagram is drawn using the values of the matrix elements greater than the threshold [34].

2.3. Research Procedure

This study went through several stages: beginning with data collection, then the standardization of terms, content analysis, survey collection, and, finally, the derivation of core defects (Figure 1). In the first stage, previously studied embedded software defects and critical factors are collected without omission. Second, the terms are standardized to eliminate the errors caused by differences in classification, as the terms used by researchers did not match. Third, the bootstrapping content analysis technique is used (Table 3). Fourth, the opinions of experts are collected through questionnaires, and the cause-and-effect relationships among defects are analyzed using the DEMATEL technique. Finally, the core defects are derived by analyzing the resulting cause-and-effect relationship diagram.

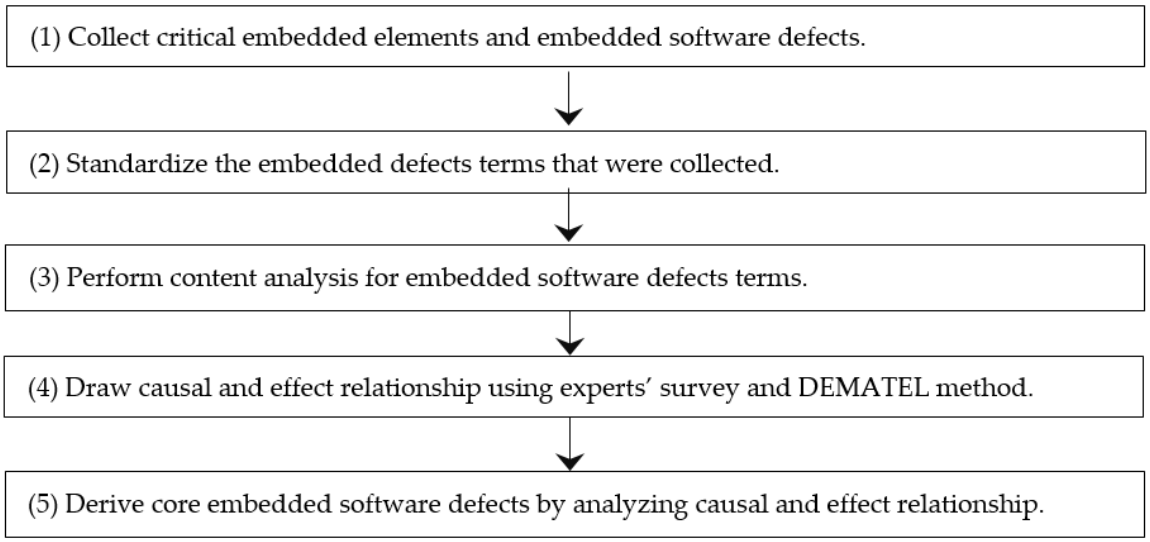

Figure 1. Research procedure for classification of collected defects and derivation of core defects.

2.4. Materials

2.4.1. Collected Critical Embedded Elements and Embedded Software Defects

Only pure software defects and embedded hardware-controlling software defects were collected for this study. It must be noted that hardware-controlling defects were excluded from the study. Using the pure software defects that were previously studied (Table 1) and classified (Table 2), embedded software defects were collected, as shown in Table 5.

Table 5. Embedded software defects and sources.

Researcher	Embedded Software Defects	Researcher	Embedded Software Defects
Barr [9]	Race condition Non-reentrant function Missing volatile keyword Stack overflow Heap fragmentation	Barr [10]	Memory leak Deadlock Priority inversion Incorrect priority assignment

Table 5. *Cont.*

Researcher	Embedded Software Defects	Researcher	Embedded Software Defects
Ji and Bao [35]	Initialize Input Interface Output Control Fault detection Fault handles Performance	Noergaard [36]	Managing data: serial and parallel I/O Interfacing the I/O components Device driver Flash memory Multitasking and process management Memory management I/O and file system management
Choi et al. [37]	Task management Inter-task communication Time management Interrupt Signal processing I/O management Memory management Networking File system	Jung et al. [14]	Types Declarations and definitions Pointer type conversion Arithmetic type conversion Expressions Control flow Control statement expressions Switch statements Functions
Hagar [12]	Data computation bug Structural logic flow Long duration control Logic and control law Data Computation Software(S/W)-to-Hardware(H/W) interface H/W-to-S/W interface S/W-system fault tolerance S/W error recovery H/W to S/W communications bug Time-related Human interface	Seo and Choi [38]	Memory Timer I/O device Task management Exception handling Inter-task communication Virtual memory management Physical memory management Time management Interrupt handling I/O management (i.e., device driver I/O) Networking File system
Sung et al. [39]	Task management Inter-task management Time management Interrupt/signal/exception handling Memory management I/O management Networking File system I/O device Timer Hardware initialization	Rodriguez -Dapena [40]	Calculations faults Data faults Internal interface faults Logic faults Control flow faults Interface between components Control flow between components H/W to S/W interface faults H/W to S/W interface User interface faults
Seo [16]	S/W-to-H/W interface S/W-to-S/W interface	Lee [18]	Exception handling
Lutz [11]	Process flow Program fault Interface specification Internal faults Interface faults Functional faults	Bennett and Wennberg [15]	Internal faults Interface faults Function faults Operating faults Condition faults Behavior faults
Sung [41]	Kernel Interface Task management Inter-task communication Time management Interrupt/exception handling Memory management I/O management Networking File system Hardware interface	Lee [22]	Assignment Checking Interface Algorithm
		Lee et al. [20]; Lee and Park [21]	Time out Data violation Complete with delay Error without effect Exception

Table 5. *Cont.*

Researcher	Embedded Software Defects		Researcher	Embedded Software Defects	
Jung [42]	Control logic		Jones [43]	Getting bored and running	Run-time environment (e.g., stack and heap allocation, memory models, etc.)
	Sensor	Missing S/W logic between sensor and system		Knocking off the obvious mistakes	Initialization
		Wrong alalog/digital conversion			Pointer dereferencing
	Key	Wrong A/D conversion table, Key alone event		Background/ foreground issues	Arithmetic errors
	LCD panel	Display error			Reentrancy
	Indicator	Indication error			Atomicity
		Buzzer			Interrupt response times
	Motor actuator			Timing related	Resource allocation mistake
	Analog/digital (A/D) conversion error				Priority/schedule issues
	Digital/analog (D/A) conversion error				Deadlocks
					Priority inversion
					Race conditions
Durães and Madeira [44]; Cotroneo et al. [19]	Missing variable initialization Missing variable assignment with a value Missing variable assignment with an expression The incorrect value assigned to a variable Missing function call Missing IF construct + statements Missing IF construct + statements + ELSE construct Missing small and localized part of the algorithm Missing IF construct around statements Missing AND in expression used as branch condition Missing OR in expression used as branch condition Wrong variable used in the parameter of function call Wrong arithmetic expression in function call parameter		Lee et al. [13]	Input data handling logic	Aanalog/digital sampling Aanalog/digital conversion Fail-safe
				Control logic	Interrupt Expression Data processing Branch control Loop control
				Output data Handling logic	Output port set Abort output (Incorrect time, feedback control error) Fail-safe
			YN Choi [17]	Memory allocation	Leakage Zero allocation Fail allocation
				Memory access	Null pointer access Free pointer access Invalid pointer access Outbound access Collision
				Memory free	Illegal free Null pointer free Duplicate free

2.4.2. Standardization of Terms

As the terms of defects studied by each researcher in Tables 2 and 5 are not consistent, this study standardized the terms of defects. Standardization was discussed with four embedded software experts (as shown in Table 6) who helped classify representative words based on the defects classified in the previous studies in Table 2. The standardized terms of defects that were identified are shown in Table 7.

Table 6. Four embedded software experts involved in terms standardization.

Experts	Embedded Software Expert		
	Field of Embedded Software Development	**Number of Years of Embedded Software Experience**	**Experts Class or Certification**
Experts 1	Mobile, Internet of thing	20 years	IT Auditor
Experts 2	Internet Cable TV, IoT, Device driver	10 years	Professional engineer
Experts 3	Mobile, industrial device control	8 years	Top engineer
Experts 4	Industrial control	8.5 years	Professional engineer

Table 7. Standardized embedded software defect terms.

Code	Embedded Software Defects	Code	Embedded Software Defects
1-1	Data access	4-1	Wrong interrupts
1-2	Shared memory	4-2	Incorrect subroutine called
1-3	Data violation	4-3	Nonexistent subroutine call
1-4	Data boundary error	4-4	Wrong parameter
1-5	Type mismatch	4-5	Inter-task communication
1-6	Save storage data	4-6	Internal interface
1-7	Flash memory	4-7	Module interface
1-8	Memory initialization	4-8	Incorrect API usage
1-9	Memory management	4-9	Wrong protocol
1-10	Memory access	4-10	Software architecture
1-11	Resource leaks	4-11	Exception handling
1-12	Memory free error	4-12	None sensor logic
1-13	Memory overflow error	5-1	Missing computation
1-14	Memory violation error	5-2	Incorrect operand and operator
2-1	Wrong H/W interface	5-3	Incorrect parenthesis
2-2	I/O devices	5-4	Round and truncate
2-3	User interfaces	5-5	Sign convention
2-4	External interface	5-6	Divide by zero
2-5	Send and receive packets error	5-7	Arithmetic overflow and underflow
2-6	Networking	6-1	Wrong logic
2-7	Input value error	6-2	Non-reentrant function
2-8	Output signal	6-3	Wrong objects
2-9	Data I/O process	6-4	Wrong relationship
2-10	Incorrect sensor data	6-5	Incorrect return
3-1	Optimization	6-6	Logic error
3-2	Time out	7-1	Infinite loops
3-3	Time fault causes data loss	7-2	If and case statements
3-4	Complete with delay	7-3	Check variables
3-5	Time delay	7-4	Serialization
3-6	Feedback control error	7-5	Deadlock
3-7	Set time and read	7-6	Concurrent processing
3-8	Time management	7-7	Task management
		7-8	Recursion

2.4.3. Embedded Software Defects via Content Analysis

Using the content analysis procedures (as shown in Table 3), the researchers and collaborators (as shown in Table 6) classified defects in Table 7 and used the reliability table in Table 4 to derive matching classifications. This process was repeated several times to extract the 12 embedded software defects shown in Table 8. The categorization index, a ratio of the agreed value located on the diagonal line of Figure 2, was used to confirm the reliability of the agreed-upon classifications. The classification index was evaluated at about 96%, with 64 of the 66 defects agreed upon by the researchers and experts, qualifying it as "excellent."

Table 8. Derived final embedded software defects.

Code	Embedded Software Defects	Definition	Sub-Defects
E1	Wrong logic	Control logic and calculation	Control flow, if, case, loop statements, divided by zero
E2	Wrong function	Function itself defects	Non-reentrant function, incorrect objects
E3	Task management	Concurrent processing error	Deadlock, race condition, task management
E4	Exception handling	Device driver exception handle error	Software exception handling excluding device driver error
E5	Internal software interface	Communication error between software	Internal interface, inconsistent module interface, wrong parameter
E6	External interface	Communication error with the external system	Networking, send and receive packet error, human interface
E7	Device driver	Hardware control device driver	I/O device, I/O port process, I/O device status
E8	Hardware interrupt	The processing routine for hardware interrupt	Non interrupt routine, incorrect interrupt routine, process error
E9	Timing error	Defects that cannot complete in time	Time out, time delay, feedback control error, set time and read time
E10	Data, shared memory	Data and static memory	Data definition, data access, shared memory
E11	Dynamic memory	Defect using dynamic memory	Memory initialization, memory management, resource leaks, memory overflow
E12	Flash memory and file system	Data storage device	Flash memory, storage data save

Experts / Researcher	Wrong logic	~~Calculation~~	Wrong function	Task manage-ment	Exception handling	~~Inter-task comm~~ Internal S/W interface	External ~~communicat ion~~ interface	~~H/W~~ Device driver	H/W interrupt process	Timing error	Data & shared memory	Dynamic memory	Flash memory & File system
Wrong logic	7-1, 7-2, 7-3, 3-1, 6-1, 6-6, 5-1, 5-2, 5-3, 5-4, 5-5, 5-6, 5-7, ~~4-11, 7-4~~												
~~Calculation~~		~~5-1, 5-2, 5-3, 5-4, 5-5, 5-6, 5-7~~											
Wrong function			6-2, 6-3, 6-5. 4-3										
Task management	~~3-6~~			7-5, 7-6, 7-7, 7-8									
Exception handling					7-4, 4-11								
~~Inter-task comm~~ Internal S/W interface						6-4, 4-4, 4-5. 4-6. 4-7, 4-8, 4-10							
External ~~communicati on~~ interface							4-9, 2-4, 2-5, 2-6						
~~H/W~~ Device driver								2-1, 2-2, 2-3, 2-7, 2-8, 2-9					
H/W interrupt process									~~4~~-1, 4-2, 4-12				
Timing error										3-2, 3-3, 3-4, 3-5, 3-6, 3-7, 3-8			
~~Data access~~ Data & shared memory											2-10, 1-1. 1-2, 1-3, 1-4, 1-5		
Dynamic memory												1-8, 1-9, 1-10, 1-11, 1-12, 1-13, 1-14	
Flash memory & File system													1-6, , 1-7

Figure 2. The final reliability table that was agreed upon between the researchers and collaborators based on the standardization of terms shown in Table 7.

3. Results

3.1. Derived 12 Embedded Software Defects

The 12 embedded software defect classes and their sub-defects were generated after analyzing and standardizing the numerous terms collected. This defect classification includes all the collected defects, and targeting the defects summarily listed here may mitigate the risk that a defect will remain untested. Next, using the DEMATEL method, this study determined the relationships between embedded software defects to derive core defects

3.2. Expert Opinions on the Influence Relationships of Embedded Software Defects

The opinions of 16 experts (with an average of 9.5 years of embedded software development experience) were collected using a survey to analyze the impact of the 12 identified defects. These experts are professional engineers, top engineers, and information technology (IT) auditors, with 6 to 20 years of embedded software experience, as shown in Table 9, along with their specific fields and survey analysis results. The impact values of the 12 defects were collected from these experts with values ranging from zero to four (zero—no impact, one—low impact, two—normal impact, three—high impact, four—very high impact). Cronbach's α was used to measure the reliability of the survey. Its value was 0.906 using the SPSS tool and the Cronbach's α, as shown in Table 9.

Table 9. Summary of Cronbach's α analysis results for survey of embedded expert respondents.

No.	Embedded Expert Respondents			Survey Analysis Result of Cronbach's α			
	Field of Embedded Software Development	Number of Years of Embedded Software Experience	Respondent Class or Certification	Scale Average if This Item was Deleted	Scale Distribution if This Item was Deleted	Modified Full Correlation	Cronbach's α if this Item was Deleted
R1	IP CCTV, IoT, Device driver	10	P.E. *	26.639	98.358	0.765	0.895
R2	Device driver	6.5	P.E. *	26.882	99.615	0.736	0.897
R3	Industrial device control	6	P.E. *	27.778	103.559	0.312	0.911
R4	Mobile, IoT	20	IT auditor	27.333	101.231	0.534	0.902
R5	Mobile, IoT	9.5	IT auditor	27.222	99.447	0.676	0.898
R6	Mobile	10.5	IT auditor	27.132	97.346	0.671	0.897
R7	Mobile	14	P.E. *	27.681	102.680	0.584	0.901
R8	Mobile, IoT	12	P.E. *	27.208	99.649	0.546	0.902
R9	IoT	8	Top engineer	27.111	98.155	0.632	0.899
R10	Network cam,	7	P.E. *	26.757	98.843	0.717	0.897
R11	Mobile, industrial device control	8	Top engineer	27.819	101.743	0.478	0.904
R12	Industrial control	8.5	P.E. *	27.882	98.748	0.569	0.901
R13	Mobile	10	P.E. *	27.882	98.748	0.569	0.901
R14	Intrusion Prevention System	6	P.E. *	27.569	97.939	0.477	0.906
R15	IoT	5	Top engineer	26.604	98.507	0.693	0.897
R16	Home automation	5	Top engineer	27.167	96.252	0.616	0.900
	Average	9.125	N/A	N/A	N/A	N/A	0.906

* P.E.: Professional engineer.

3.3. DEMATEL Analysis of Expert Opinions

The DEMATEL method was applied to the questionnaire in stages to analyze the impact relationships of defects. First, for the collected questionnaire values, the arithmetic mean was calculated using equation (1), and this was used to generate the generalized matrix (A; Table 10). Second, to normalize the generalized matrix (A), the maximum value was calculated using equation (2) and applied to equation (3), thus deriving a normalized matrix (N). Third, the TRM (T) was calculated (Table 11) by multiplying N by the inverse matrix of the unit matrix (equation 4). Fourth, this study calculated (equation 5) for the sum of columns (D) and the sum of the rows (R), (D+R), and (D-R) factors in the TRM (T) of Table 12. Fifth, the threshold value was calculated using equation (7) to get a value of 0.4928. Values smaller than the threshold values in matrix (T) were identified as having no

impact, whereas larger values have an impact. Finally, the factor (D-R) in Table 12 is set on the *y*-axis, while the factor (D+R) is set on the *x*-axis. These are then used to draw the impact relationship map.

Table 10. Generalized cause and effect matrix (*A*).

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12
E1	0.0000	2.3125	2.6250	2.3750	1.6875	1.6250	2.4375	2.1875	2.6250	2.5000	2.6250	2.5000
E2	2.3125	0.0000	2.7500	2.3750	2.3125	2.1875	2.1875	2.1250	2.1875	2.2500	2.3125	1.6250
E3	1.9375	1.7500	0.0000	2.1250	2.3750	2.4375	2.2500	2.5625	2.6250	2.3125	2.3125	1.6875
E4	2.3125	1.9375	2.3125	0.0000	1.7500	1.9375	1.7500	1.8125	2.0625	1.7500	1.6250	1.3750
E5	1.9375	2.0625	2.3750	1.9375	0.0000	2.1875	2.0625	1.8750	2.2500	1.3750	1.6250	1.5000
E6	1.6250	1.3750	1.7500	1.6875	1.7500	0.0000	1.8750	2.0625	2.8125	1.4375	1.3750	0.8750
E7	1.5625	1.5000	1.8125	1.6250	1.6875	2.1875	0.0000	2.3125	2.5000	1.8125	1.8750	1.2500
E8	1.3750	1.2500	1.9375	1.3125	1.6250	2.1875	2.7500	0.0000	2.5000	1.6875	1.6875	1.4375
E9	1.6250	1.6875	2.3750	1.7500	2.0000	2.7500	2.4375	2.3750	0.0000	1.6875	1.3750	1.3750
E10	2.3125	2.3750	2.3125	2.0000	2.0000	1.8750	2.0000	1.9375	1.9375	0.0000	2.3125	1.6875
E11	2.2500	2.3125	2.1875	2.1250	2.0000	1.7500	2.0000	1.8750	2.0625	2.3125	0.0000	1.6250
E12	1.6875	1.6250	2.1250	1.3750	1.6875	1.7500	2.0625	2.3125	1.8125	2.1250	1.9375	0.0000

Table 11. Total cause and effect matrix (*T*).

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12
E1	0.4742	0.5415	0.6444	0.5562	0.5387	0.5825	0.6266	0.6103	0.6662	0.5715	0.5711	0.4786
E2	0.5421	0.4435	0.6307	0.5416	0.5444	0.5846	0.6007	0.5907	0.6343	0.5467	0.5448	0.4360
E3	0.5205	0.4992	0.5229	0.5239	0.5378	0.5845	0.5939	0.5961	0.6390	0.5395	0.5353	0.4306
E4	0.4732	0.4474	0.5365	0.3879	0.4563	0.5006	0.5081	0.5035	0.5472	0.4602	0.4526	0.3714
E5	0.4677	0.4583	0.5472	0.4657	0.3998	0.5181	0.5274	0.5144	0.5631	0.4548	0.4595	0.3811
E6	0.4099	0.3897	0.4718	0.4103	0.4168	0.3884	0.4686	0.4684	0.5256	0.4086	0.4031	0.3210
E7	0.4337	0.4189	0.5034	0.4336	0.4405	0.4951	0.4291	0.5054	0.5459	0.4477	0.4462	0.3549
E8	0.4189	0.4024	0.4980	0.4148	0.4305	0.4869	0.5181	0.4140	0.5368	0.4355	0.4319	0.3545
E9	0.4573	0.4456	0.5472	0.4594	0.4730	0.5383	0.5412	0.5323	0.4838	0.4653	0.4514	0.3766
E10	0.5112	0.4991	0.5803	0.4982	0.5026	0.5394	0.5591	0.5495	0.5879	0.4345	0.5140	0.4130
E11	0.5048	0.4927	0.5711	0.4980	0.4981	0.5304	0.5540	0.5423	0.5865	0.5130	0.4262	0.4072
E12	0.4466	0.4317	0.5238	0.4335	0.4490	0.4888	0.5134	0.5150	0.5325	0.4679	0.4581	0.3156

Table 12. Results of the cause and effect analysis of embedded software defects.

Code	Defects	D	R	D+R	D-R
E1	Wrong logic	6.86	5.66	12.52	1.20
E2	Wrong function	6.64	5.47	12.11	1.17
E3	Task management	6.52	6.58	13.10	−0.05
E4	Exception handling	5.64	5.62	11.27	0.02
E5	Internal software interface	5.76	5.69	11.44	0.07
E6	External interface	5.08	6.24	11.32	−1.16
E7	Device driver	5.45	6.44	11.89	−0.99
E8	Hardware interrupt	5.34	6.34	11.68	−1.00
E9	Timing error	5.77	6.85	12.62	−1.08
E10	Data, shared memory	6.19	5.75	11.93	0.44
E11	Dynamic memory	6.12	5.69	11.82	0.43
E12	Flash memory and file system	5.58	4.64	10.22	0.94

3.4. Influence Analysis between Embedded Software Defects

The DEMATEL method determined the degree of a defect's influence power as each defect related to each other, enabling this study to plot an IRM, as shown in Figure 3. The D column lists are the sum of rows, and the R column lists are the sum of columns. The D value numerically expresses the degree to which one defect affects other defects, while the R-value expresses the degree to which one defect is affected by other defects. D+R is the sum of the values affecting other defects and the values affected by other defects. The D+R value is useful for identifying the total value of defects. On the other hand, D-R is the difference of values that affect other defects and the value affected by other defects. The larger the value of D-R, the greater the influencing power of the defect, while the smaller its value, the more it is affected by other defects. Therefore, defects with positive D-R values are cause defects and belong to the cause group, while defects with negative D-R values are effect

defects and belong to the effect defect group [25]. Additionally, the affecting defect, or the cause defect, should be tested first because it affects other defects. Meanwhile, the affected defect should be tested later as it is affected by other defects [30]. Therefore, defects with higher D-R values should be tested first, whereas defects with lower D-R values should be tested later.

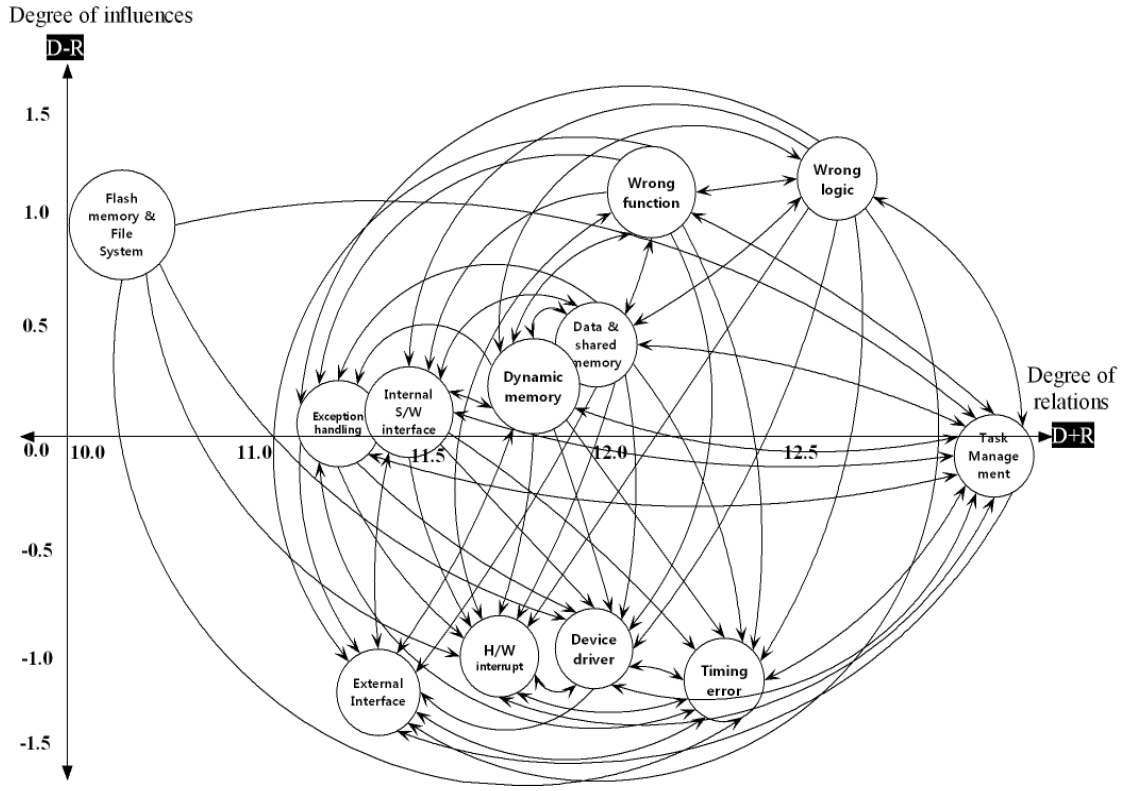

Figure 3. The impact relationship map of embedded software defects.

As shown in Table 13, the D-R values of the following defects are positive and should be tested first: wrong logic (E9), wrong function (E10), flash memory and file system defects (E6), data and shared memory (E4), dynamic memory (E5), internal software interface (E7), and exception handling (E12). Meanwhile, the D-R values of the following defects are negative and should be tested later: task management (E11), device driver (E1), hardware interrupt (E2), timing error (E3), and the external interface (E8) defects.

As defined at the beginning of the paper, core defects refer to defects that increasingly get more severe due to other defects. Based on this, the five core defects that have been identified in the effect group with negative D-R values and different characteristics are the following: external interface (E6), timing error (E9), hardware interrupt (E8), device driver (E7), and task management (E3). Minimizing these defects is vital, so tests that are appropriate for each defect characteristic should be performed. The characteristics of the five core defects that were derived are as follows: the primary core defects are defects with the smallest D-R value, which is an external interface fault, including network, serial port, and human interface. For example, if there are defects in the human interface, other functions can still work in a particular way and cause problems even if the user requests for the desired function. It can be understood as the most important fault as it can lead to serious problems due to incorrect operation if commands from an external system are incorrectly received. The second important defect is the hardware interrupt defect, which includes operations like dividing by zero, overflow, underflow, etc. If defects that interrupt processing occur, serious problems may follow. The third important defect is the device driver, providing the interface to control the hardware. Defects occurring in the device driver are essential to note because they prevent users from predicting how the embedded system will

operate. The fourth important defect is the timing error. Embedded systems can be directly linked to human life, such as automobile autonomous navigation systems, automatic navigation systems in aviation, nuclear power plant control systems, and missile control devices in the defense industry. If an immediate response function times out, unpredictable consequences may occur. The last important defect is the task management defect since tasks may not be performed normally due to deadlock, race condition, etc.

Table 13. Embedded software defects sorted with D+R and D-R.

Code	Defects Sorted by D+R	D+R	Code	Defects Sorted by D-R	D-R
E3	Task management	13.1	E1	Wrong logic	1.2
E9	Timing error	12.62	E2	Wrong function	1.17
E1	Wrong logic	12.52	E12	Flash memory and file system	0.94
E2	Wrong function	12.11	E10	Data, shared memory	0.44
E10	Data, shared memory	11.93	E11	Dynamic memory	0.43
E7	Device driver	11.89	E5	Internal software interface	0.07
E11	Dynamic memory	11.82	E4	Exception handling	0.02
E8	Hardware interrupt	11.68	E3	Task management	−0.05
E5	An internal software interface	11.44	E7	Device driver	−0.99
E6	External interface	11.32	E8	Hardware interrupt	−1
E4	Exception handling	11.27	E9	Timing error	−1.08
E12	Flash memory and file system	10.22	E6	External interface	−1.16

In a comprehensive interpretation of core defects, this study found that embedded systems should be executed robustly without being affected by external systems and environments, and that interrupt should be handled correctly. They should then be implemented to respond to different types of hardware. The desired function must be performed within a limited time, ensuring that the original function is executed faithfully without damaging other task types. Therefore, applying a test method suitable for such characteristics would be the best way to minimize defects.

3.5. Validation with Embedded Software Developer Experts

This study collected six critical defects—considered the most important of the 12 embedded defects listed in Table 8—from 10 embedded software development experts to confirm the reliability of the study results. Table 14 illustrates that although some experts suggested that logic defects, exception handling, data, and dynamic memory defects are also important, the common opinion is that hardware interrupts, external software interface, timing error, device drivers, and task management defects are the biggest impediments to proper system functioning. When looking at the important defects derived from experts and the core defects derived from this study, there are only slight differences, with the rest being approximately identical.

Table 14. Opinions of embedded software development experts on important defects.

Code	Defects	Opinions of 10 Embedded Software Experts										Rank
		1	2	3	4	5	6	7	8	9	10	
E1	Device driver	6	4	4		1	2	3	1		1	4
E2	H/W interrupt	4	2	2	2	3	1	4	5		4	1
E3	Timing error	5	3		5	4	5	2	3	6	5	3
E4	Data and shared memory			6	5				5			
E5	Dynamic memory	2	5					6				
E6	Flash memory and file system	1				6				5	2	
E7	Internal software interface									1		
E8	External interface		6	3	3	5	3	1	2	2	3	2
E9	Wrong logic		1				6			3	6	
E10	Wrong function								4	4		
E11	Task management	3		5	1		4	5				5
E12	Exception handling			4	4	2			6			6

3.6. The Difference between Previous Studies and This Study

Current embedded software defect research only includes specific areas that researchers consider essential, as shown in Table 5. If defects are tested according to previous studies, there may be defects that are not tested, which may cause failures. For this reason, the present study collected embedded software defects worldwide, classified them as MECE and organized them into 12 categories and sub-defects to solve this problem. Therefore, if 12 categories and sub-defects are used and tested, they can account for all defects tested, minimizing the failure of the embedded system.

Given that defects affect each other, problems can arise if the effect defect is tested first and the cause defect is tested later. For example, if a cause defect is found after removing the effect defects, the effect defects must be tested again, as the cause defects may affect the effect defects. Therefore, testing the cause defects first and then testing the effect defect later is a way to minimize the defect without running multiple tests [30]. In the present study, the cause defects and the effect defects were derived by analyzing the influence relationship between defects. Thus, the defect can be eliminated by testing the cause defects first and then testing the effect defects later.

Embedded software defects range from minor defects to severe defects. Naturally, more weight should be placed on severe defects than minor ones to improve the safety of embedded system. Various embedded software defects have been studied, but there is insufficient research on major defects to minimize embedded system failures. In the present study, the influence relationship between defects was analyzed to identify the major defects. The cause defects and effect defects were identified using the influence relationship between the defects. Cause defects may not cause failure by eliminating their own defects. However, even if effect defects are eliminated by their own defects, defects can be caused by cause defects. Therefore, effect defects should be intensively managed and tested more than cause defects. The effect defects derived in this study are called core defects, and it was determined that hardware-dependent defects are greatly affected by other defects. Therefore, if in-depth tests are conducted on the core defects derived in this study, the failure of the embedded system can be minimized.

4. Conclusions

This study was able to derive 12 defect categories and sub-defects using the content analysis technique, draw the cause and effect relationship between embedded software defects, and derive core defects using the DEMATEL method. After studying the data yielded throughout the different stages of the study's analyses, the results show that the core embedded software defects were the external interface defect, the hardware interrupt defect, device driver defect, timing error, and task management defect.

What this study does is integrate and organize pure software defects and embedded software defects from around the world, opening avenues for other researchers to improve software quality. This study also helps mitigate the risks that come from critical defects that might not have been tested. Moreover, the impact relationships between defects can be better mapped through the diagrams presented here. Lastly, using the cause and effect relationships, this study constructed a basis for estimating defect weights. Future studies may validate and use them as criteria for targeting embedded software defects.

There are also many industrial applications for this study. First, by eliminating the time required to collect and classify the defects, one may immediately inspect and target any defects that may be present. Second, when developing an embedded system, systems can remove defects more efficiently and effectively using a guide that orders defects by importance. Third, analyzing the priorities of the defects may facilitate a more accessible selection of the appropriate embedded software test technique. Lastly, when performing a fault injection test, this study suggests that more defects can be injected and tested in the source code where core defects are likely to occur.

In this study, embedded software defects were classified into 12 defect categories and sub-defects. Moreover, the influence relationship of defects was analyzed for each of the 12 defect categories and

classifying the defects into cause group defects and effect group defects. However, the weight of the defect was not completely calculated, and the influence relationship of the sub-defects was not analyzed. Therefore, future studies that derive the weights of sub-defects and studies that analyze the influence relationship of sub-defects to identify cause defects and effect defects at the level of sub-defects are essential. In addition, future researchers can look into how to improve the defect removal rate while conducting embedded tests (such as defect injection tests) using the defects derived in this study, compared to the existing tests.

Author Contributions: Conceptualization, S.M.H. and W.-J.K.; methodology, S.M.H. and W.-J.K.; validation, W.-J.K.; investigation, S.M.H.; resources, S.M.H. and W.-J.K.; writing—original draft preparation, S.M.H.; writing—review and editing, S.M.H. and W.-J.K.; project administration, S.M.H. and W.-J.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Leveson, N.G.; Turner, C.S. An investigation of the Therac-25 accidents. *Computer* **1993**, *26*, 18–41. [CrossRef]
2. Lann, G.L. The Ariane 5 Flight 501 failure—A case study in system engineering for computing systems. [Research Report] RR-3079. *INRIA* **1996**. Available online: https://hal.inria.fr/inria-00073613 (accessed on 2 September 2020).
3. Cousot, P.; Cousot, R. A gentle introduction to formal verification of computer systems by abstract interpretation. In *Logics and Languages for Reliability and Security*; Esparza, J., Spanfelner, B., Grumberg, O., Eds.; IOS Press: Amsterdam, The Netherlands, 2010; pp. 1–29. [CrossRef]
4. Mäntylä, M.V.; Lassenius, C. What types of defects are really discovered in code reviews? *IEEE Trans. Softw. Eng.* **2009**, *35*, 430–448. [CrossRef]
5. Huh, S.-M.; Kim, W.-J. A method to establish severity weight of defect factors for application software using ANP [Korean]. *J. KIISE* **2015**, *42*, 1349–1360. Available online: http://www.riss.kr/link?id=A101312445 (accessed on 2 September 2020). [CrossRef]
6. IEEE. IEEE standard classification for software anomalies. In *IEEE Std. 1044–2009 (Revision of IEEE Std 1044–1993)*; IEEE: New York, NY, USA, 2010; pp. 1–23. Available online: http://www.ctestlabs.org/neoacm/1044_2009.pdf (accessed on 2 September 2020).
7. Huber, J.T. A Comparison of IBM's Orthogonal Defect Classification to Hewlett Packard's Defect Origins, Types and Modes 1.0. Hewlett Packard Co. 1999. Available online: http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=ART&ObjectId=2883 (accessed on 2 September 2020).
8. IBM. Orthogonal Defect Classification v5.2 for Software Design and Code. IBM. 2013. Available online: https://researcher.watson.ibm.com/researcher/files/us-pasanth/ODC-5-2.pdf (accessed on 2 September 2020).
9. Barr, M. Five top causes of nasty embedded software bugs. *Embed. Syst. Des.* **2010**, *23*, 10–15.
10. Barr, M. Five more top causes of nasty embedded software bugs. *Embed. Syst. Des.* **2010**, *23*, 9–12.
11. Lutz, R. Analyzing software requirements errors in safety-critical, embedded systems. In Proceedings of the IEEE International Symposium on Requirements Engineering; IEEE: San Diego, CA, USA, 1993; pp. 126–133. [CrossRef]
12. Hagar, J.D. *Software Test Attacks to Break Mobile and Embedded Devices*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2013.
13. Lee, S.Y.; Jang, J.S.; Choi, K.H.; Park, S.K.; Jung, K.H.; Lee, M.H. A study of verification for embedded software [Korean]. *Ind. Eng. Manag. Sys.* **2004**, *11*, 669–676. Available online: http://www.riss.kr/link?id=A60279480 (accessed on 2 September 2020).
14. Jung, D.-H.; Ahn, S.-J.; Choi, J.-Y. Programming enhancements for embedded software development-focus on MISRA-C [Korean]. *J. KIISE Comp. Pract. Lett.* **2013**, *19*, 149–152. Available online: http://www.riss.kr/link?id=A99686225 (accessed on 2 September 2020).
15. Bennett, T.; Wennberg, P. Eliminating embedded software defects prior to integration test. *J. Def. Softw. Eng. Triakis Corp.* **2005**. Available online: https://pdfs.semanticscholar.org/3070/1dcef9b58d6c167751aaeaff7c9628cf04c4.pdf (accessed on 2 October 2020).

16. Seo, J. Embedded Software Interface Test Based on the Status of System [Korean]. Ph.D. Thesis, Department of Computer Science and Engineering Graduate School, EWHA Womans University, Seoul, Korea, 2009. Available online: http://www.riss.kr/link?id=T11551362 (accessed on 2 September 2020).
17. Choi, Y.N. Automated Debugging Cooperative Method for Dynamic Memory Defects in Embedded Software System Test [Korean]. Master's Thesis, Department of Computer Science and Engineering Graduate School, EWHA Womans University, Seoul, Korea, 2010. Available online: http://dspace.ewha.ac.kr/handle/2015.oak/188271 (accessed on 2 September 2020).
18. Lee, S. Automated Method for Reliability Verification in Embedded Software System Exception Handling Test [Korean]. Master's Thesis, Department of Computer Science and Engineering Graduate School, Ewha Womans University, Seoul, Korea, 2011. Available online: https://dspace.ewha.ac.kr/handle/2015.oak/188659 (accessed on 2 September 2020).
19. Cotroneo, D.; Lanzaro, A.; Natella, R. Faultprog: Testing the accuracy of binary-level software fault injection. *IEEE T. Depend. Secure.* **2016**, *15*, 40–53. [CrossRef]
20. Lee, H.-J.; Yoon, J.-H.; Lee, K.-Y.; Lee, D.-W.; Na, J.-W. Reclassification of fault, error, and failure types for reliability verification of defense embedded systems [Korean]. *Proc. Inst. Control Robot. Syst.* **2012**, *7*, 925–932. Available online: http://www.riss.kr/link?id=A99705651 (accessed on 2 September 2020).
21. Lee, H.-J.; Park, J.-W. JTAG fault injection methodology for reliability verification of defense embedded systems [Korean]. *J. Korea Acad. Ind. Coop. Soc.* **2013**, *14*, 5123–5129. [CrossRef]
22. Lee, H.-J. Statistical JTAG Fault Injection Methodology for Reliability Verification of Aerospace Embedded Systems [Korean]. Master's Thesis, Department of Electronics and Information Engineering Korea Aerospace University, Gyeonggi, Korea, 2012. Available online: http://www.riss.kr/link?id=T12740845 (accessed on 2 September 2020).
23. Jankowicz, D. *The Easy Guide to Repertory Grids*; John Wiley and Sons: Chichester, UK, 2005.
24. Si, S.-L.; You, X.-Y.; Liu, H.-C.; Zhang, P. DEMATEL technique: A systematic review of the state-of-the-art literature on methodologies and applications. *Math. Probl. Eng.* **2018**. [CrossRef]
25. Elo, S.; Kyngäs, H. The qualitative content analysis process. *J. Adv. Nurs.* **2008**, *62*, 107–115. [CrossRef] [PubMed]
26. White, M.D.; Marsh, E.E. Content analysis: A flexible methodology. *Libr. Trends* **2006**, *55*, 22–45. Available online: http://hdl.handle.net/2142/3670 (accessed on 2 September 2020). [CrossRef]
27. Mayring, P. Qualitative content analysis. *Qual. Soc. Res.* **2000**, *1*, 1159–1176. [CrossRef]
28. Sumrit, D.; Anuntavoranich, P. Using DEMATEL method to analyze the causal relations on technological innovation capability evaluation factors in Thai technology-based firms. *Int. T. J. Eng. Manage. Appl. Sci. Technol.* **2013**, *4*, 81–103. Available online: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.411.8216&rep=rep1&type=pdf (accessed on 2 September 2020).
29. Asgharpour, M.J. *Group Decision Making and Game Theory in Operation Research*, 3rd ed.; University of Tehran Publications: Enghelab Square, Iran, 2003.
30. Seyed-Hosseini, S.M.; Safaei, N.; Asgharpour, M.J. Reprioritization of failures in a system failure mode and effects analysis by decision making trial and evaluation laboratory technique. *Reliab. Eng. Syst. Safe.* **2006**, *91*, 872–881. [CrossRef]
31. Ölçer, M.G. Developing a spreadsheet based decision support system using DEMATEL and ANP approaches. Master's Thesis, DEÜ Fen Bilimleri Enstitüsü, Turkey, 2013. Available online: http://hdl.handle.net/20.500.12397/7682 (accessed on 2 September 2020).
32. Hsu, C.-C. Evaluation criteria for blog design and analysis of causal relationships using factor analysis and DEMATEL. *Expert. Syst. Appl.* **2012**, *39*, 187–193. [CrossRef]
33. Shieh, J.-I.; Wu, H.-H.; Huang, K.-K. A DEMATEL method in identifying key success factors of hospital service quality. *Knowl. Based Syst.* **2010**, *23*, 277–282. [CrossRef]
34. Yang, Y.-P.O.; Leu, J.-D.; Tzeng, G.-H. A novel hybrid MCDM model combined with DEMATEL and ANP with applications. *Int. J. Oper. Res.* **2008**, *5*, 160–168.
35. Ji, S.; Bao, X. Research on software hazard classification. *Procedia Eng.* **2014**, *80*, 407–414. [CrossRef]
36. Noergaard, T. *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*, 1st ed.; Elsevier Inc.: Oxford, UK, 2005.

37. Choi, H.; Sung, A.; Choi, B.; Kim, J. A functionality-based evaluation model for embedded software [Korean]. *J. KIISE Softw. App.* **2005**, *32*, 1192–1205. Available online: http://www.riss.kr/link?id=A82294417 (accessed on 2 September 2020).
38. Seo, J.; Choi, B. An interface test tool based on an emulator for improving embedded software testing [Korean]. *J. KIISE Comp. Pract. Lett.* **2008**, *32*, 547–558. Available online: http://www.riss.kr/link?id=A82300048 (accessed on 2 September 2020).
39. Sung, A.; Choi, B.; Shin, S. An interface test model for hardware-dependent software and embedded OS API of the embedded system. *Comp. Stand. Inter.* **2007**, *29*, 430–443. [CrossRef]
40. Rodriguez Dapena, P. Software Safety Verification in Critical Software Intensive Systems. Ph.D. Thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, 2002. [CrossRef]
41. Sung, A. Interface based embedded software test for real-time operating system [Korean]. Ph.D. Thesis, Department of Computer Science and Engineering, Ewha Womans University, Seoul, Korea, 2007. Available online: http://www.riss.kr/link?id=T11039605 (accessed on 2 September 2020).
42. Jung, H. Failure Mode Based Test Methods for Embedded Software [Korean]. Master's Thesis, Ajou University Graduate School of Engineering, Suwon-si, Korea, 2007. Available online: http://www.riss.kr/link?id=T11077107 (accessed on 2 September 2020).
43. Jones, N. A Taxonomy of Bug Types in Embedded Systems. Stack Overflow, Embeddedgurus.Com. 2009. Available online: https://embeddedgurus.com/stack-overflow/2009/10/a-taxonomy-of-bug-types-in-embedded-systems (accessed on 2 September 2020).
44. Durães, J.A.; Madeira, H.S. Emulation of software faults: A field data study and a practical approach. *IEEE Trans. Softw. Eng.* **2006**, *32*, 849–867. [CrossRef]